Introduction to Assembly Language

Benjamin Kong — 1573684
Lora Ma ———— 1570935

ECE 212 Lab Section H11

February 24, 2020

Contents

1 Introduction

Assembly language is a low-level programming language that is converted to machine code using an assembler. Assembly language uses *mnemonics* to represent low-level machine instructions, and this makes it much more readable than machine code. It is important to note that assembly language is specific to a particular computer architecture and hence may or may not work on different systems.

The purpose of this lab was to become more familiar with assembly language using the NetBurner ColdFire microcontroller board. In order to gain experience with assembly language programming, two different programs were created using assembly. In the first part of the lab (part A), we created a program that converted an Ascii character to its hexadecimal equivalent. For example, '5' is converted to '5' and 'B' is converted to '11.' However, if a character without a hexadecimal equivalent is entered, such as 'J,' then an error code results.

For the second part of the lab (part B), we created a program that converts an Ascii letter into its uppercase or lowercase equivalent. For example, 'a' is converted to 'A' while 'E' is converted to 'e.' If an invalid Ascii character is input (such as an ampersand '&,' for example), an error code results.

2 Design

2.1 Part A

First, in our program we initialize address registers $a2$ and $a3$ with addresses and store the number of times the program will loop in a data register. Then, we check if at any point in our program the enter key is pressed. If it is pressed, the program exits. Otherwise, we check if our character is valid through a series of tests. First, we check if the given value is less than the hex value '0.' If it is, then it is an invalid character and the character is replaced with an error code. If the value is greater than the hex value '0,' then we compare the value to the hex value '9.' If the value is greater than '9,' we further test to see if the value is a character between the hex value 'A' and the hex value 'F.' If the value is smaller than the hex value 'A,' then it is an invalid character. If the value is greater than 'F', we check if the character is between the hex value 'a' and 'f.' If the value is less than the hex value 'a' or greater than the hex value 'f,' it is an invalid character. After we've established whether or not the character is valid, we convert the value to its hexadecimal equivalent.

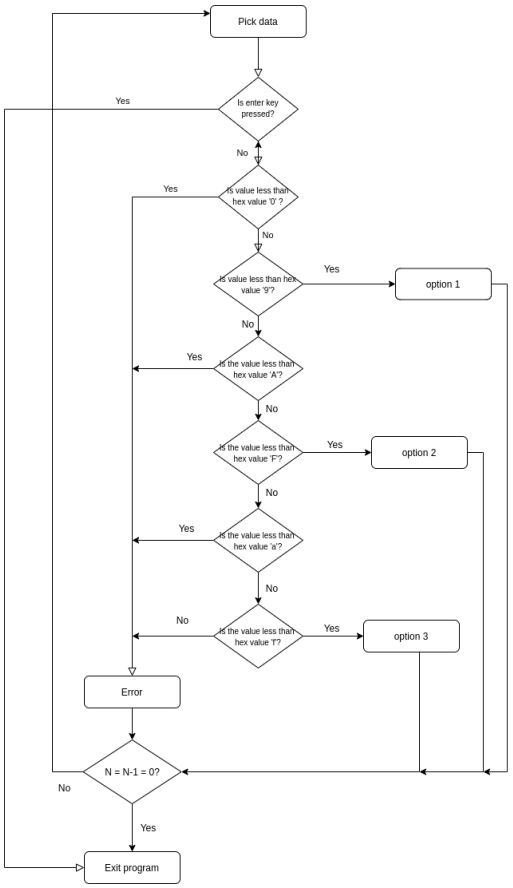

Figure 1: Flow chart for part A.

2.2 Part B

First, we initialize address registers $a2$ and $a3$ with addresses and store the number of times the program will loop in a data register. Then, we check if at any point in our program the enter key is pressed. If it is pressed, the program exits. Otherwise, we check if our character is valid through a series of tests. First, we check if the character is between the hex value 'A' and the hex value 'Z.' If it is less than 'A,' then it is not a valid character. If it is greater than 'Z,' we further test to see if the character is between the hex value 'a' and the hex value 'z.' If the character is either less than 'a' or greater than 'z,' it is an invalid character. If we have an invalid character, the character is replaced with an error code and we move on to the next value. If we have a valid character, we convert the letter to its lowercase/uppercase equivalent.

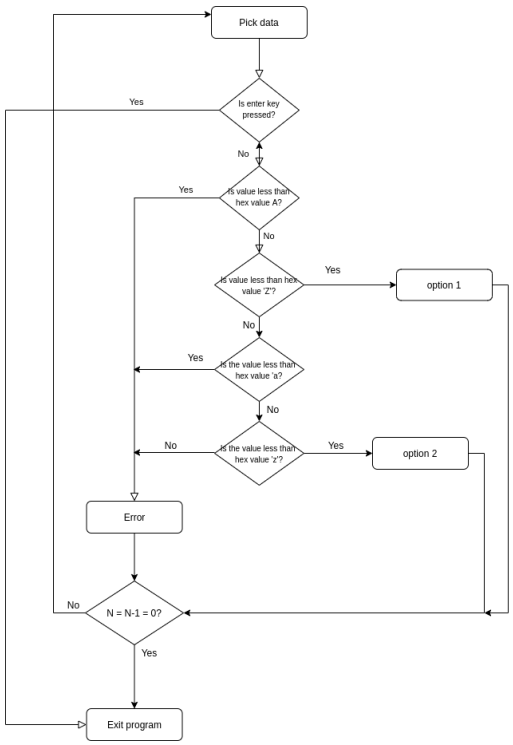

Figure 2: Flow chart for part B.

3 Testing

3.1 Part A

If properly implemented, part A converts valid Ascii characters that are hexadecimal values into their decimal equivalent. In order to test our implementation of part A, we first ran the code with the manual inputs. For example, we input various valid and invalid Ascii characters such as 'A' (which would be considered valid and should result in an output of '10') and 'J' (which would be considered invalid and should result in the error code). Once we were confident that our code was working, we moved on to the test cases provided on eClass. Upon running these test cases under the supervision of a TA, our program was confirmed to be in working order and we moved on to the next part of the lab, part B. For a screenshot of the MTTY terminal, please take a look at the appendix.

3.2 Part B

If properly implemented, part B converts valid Ascii characters into their lowercase or uppercase counterparts. For example, 'A' would be converted to 'a,' 'j' would be converted to 'J,' and '%' would result in an error code since '%' doesn't have an uppercase or lowercase equivalent. After finishing our code for part B, we again tested manual inputs to check that our program functioned as expected. Once we were confident that our implementation was correct, we moved on to the test cases provided on eClass. Using these test cases, a TA confirmed that our code functioned correctly and we cleaned up our workstation. For a screenshot of the MTTY terminal, please take a look at the appendix.

4 Questions

Q1). *What happens when there is no exit code '0x0D' provided in the initialization process? Would it cause a problem? Why or why not?*

This would not be an issue since input is limited to 100 inputs. This is because the program decrements the value stored at data register $d7$ each time a character is input and the program can only loop if the value at $d7$ is not zero. As soon as 100 inputs are present and processed, the value of $d7$ will be zero and the program will exit.

Q2). *How can our code be modified to provide a variable address range? For example, what if I only wanted to convert the first 10 data entries?*

We could initialize another variable representing the address range, say 10, at $d6$. For example, we could add the line

```
1  move.l #10, %d6
```

at the beginning and decrement this each time a data entry is converted. If the value at $d6$ is zero, then we stop converting data entries.

5 Conclusion

The purpose of this lab was to become more familiar with assembly language using the NetBurner Coldfire microcontroller board. In order to do this, we completed two parts: part A, which involved converting valid Ascii characters into the equivalent hexadecimal value; and part B, which involved converting valid uppercase Ascii characters into their lowercase equivalent and vice versa for valid lowercase Ascii characters.

Overall, the lab was a success: both parts of the lab were completed in a timely fashion and our understanding of assembly language was increased. Both part A and part B of the lab were shown to work with the test cases provided on eClass. In the future, we would like to become even more proficient with assembly language and we look forwards to completing future labs. :)

6 Appendix

6.1 Part A MTTY Screenshots

```
The conversion is starting at 0x2300000:
Ascii Character = 0 Decimal value = 0
Ascii Character = 1 Decimal value = 1
Ascii Character = 2 Decimal value = 2
Ascii Character = 3 Decimal value = 3
Ascii Character = 4 Decimal value = 4
Ascii Character = 5 Decimal value = 5
Ascii Character = 6 Decimal value = 6
Ascii Character = 7 Decimal value = 7
Ascii Character = 8 Decimal value = 8
Ascii Character = 9 Decimal value = 9
Ascii Character = A Decimal value = 10
Ascii Character = B Decimal value = 11
Ascii Character = C Decimal value = 12
Ascii Character = D Decimal value = 13
Ascii Character = E Decimal value = 14
Ascii Character = F Decimal value = 15
Ascii Character = a Decimal value = 10
Ascii Character = b Decimal value = 11
Ascii Character = c Decimal value = 12
Ascii Character = d Decimal value = 13
Ascii Character = e Decimal value = 14
Ascii Character = f Decimal value = 15
Welcome to lab1 test program, please select
Press 1 to test part a
Press 2 to test part b
```

Figure 3: Screenshot of MTTY output for part A using DataStorage.s.

```
The conversion is starting at 0x2300000:
Ascii Character = B Decimal value = 11
Ascii Character = ¥ Decimal value = -1
Ascii Character = ┼ Decimal value = -1
Ascii Character =  Decimal value = -1
Ascii Character =  Decimal value = -1
Ascii Character = / Decimal value = -1
Ascii Character = 5 Decimal value = 5
Ascii Character = : Decimal value = -1
Ascii Character = @ Decimal value = -1
Ascii Character = D Decimal value = 13
Ascii Character = G Decimal value = -1
Ascii Character = P Decimal value = -1
Ascii Character = ` Decimal value = -1
Ascii Character = d Decimal value = 13
Ascii Character = g Decimal value = -1
Welcome to lab1 test program, please select
Press 1 to test part a
Press 2 to test part b
```

Figure 4: Screenshot of MTTY output for part A using DataStorage1.s.

6.2 Part B MTTY Screenshots

```
The conversion is starting at 0x2300000:
Ascii Character = 0 Upper or lower case equivalent =
Ascii Character = 1 Upper or lower case equivalent =
Ascii Character = 2 Upper or lower case equivalent =
Ascii Character = 3 Upper or lower case equivalent =
Ascii Character = 4 Upper or lower case equivalent =
Ascii Character = 5 Upper or lower case equivalent =
Ascii Character = 6 Upper or lower case equivalent =
Ascii Character = 7 Upper or lower case equivalent =
Ascii Character = 8 Upper or lower case equivalent =
Ascii Character = 9 Upper or lower case equivalent =
Ascii Character = A Upper or lower case equivalent = a
Ascii Character = B Upper or lower case equivalent = b
Ascii Character = C Upper or lower case equivalent = c
Ascii Character = D Upper or lower case equivalent = d
Ascii Character = E Upper or lower case equivalent = e
Ascii Character = F Upper or lower case equivalent = f
Ascii Character = a Upper or lower case equivalent = A
Ascii Character = b Upper or lower case equivalent = B
Ascii Character = c Upper or lower case equivalent = C
Ascii Character = d Upper or lower case equivalent = D
Ascii Character = e Upper or lower case equivalent = E
Ascii Character = f Upper or lower case equivalent = F
Welcome to lab1 test program, please select
Press 1 to test part a
Press 2 to test part b
```

Figure 5: Screenshot of MTTY output for part B using DataStorage.s.

```
The conversion is starting at 0x2300000:
Ascii Character = B Upper or lower case equivalent = b
Ascii Character = ¥ Upper or lower case equivalent =
Ascii Character = ┼ Upper or lower case equivalent =
Ascii Character =  Upper or lower case equivalent =
Ascii Character =   Upper or lower case equivalent =
Ascii Character = / Upper or lower case equivalent =
Ascii Character = 5 Upper or lower case equivalent =
Ascii Character = : Upper or lower case equivalent =
Ascii Character = @ Upper or lower case equivalent =
Ascii Character = D Upper or lower case equivalent = d
Ascii Character = G Upper or lower case equivalent = g
Ascii Character = P Upper or lower case equivalent = p
Ascii Character = ` Upper or lower case equivalent =
Ascii Character = d Upper or lower case equivalent = D
Ascii Character = g Upper or lower case equivalent = G
Welcome to lab1 test program, please select
Press 1 to test part a
Press 2 to test part b
```

Figure 6: Screenshot of MTTY output for part B using DataStorage1.s.

6.3 Part A Assembler Code

```
/* DO NOT MODIFY THIS ------------------------------------------*/
.text

.global AssemblyProgram

AssemblyProgram:
lea       -40(%a7),%a7 /*Backing up data and address registers */
movem.l %d2-%d7/%a2-%a5,(%a7)
/*-----------------------------------------------------------------*/

/*****************************************************************/
/* General Information *******************************************/
/* File Name: Lab1a.s ********************************************/
```

```
14  /* Names of Students: Lora Ma and Benjamin Kong                  **/
15  /* Date: 3 February 2020                                        **/
16  /* General Description: Converts Ascii characters to its        **/
17  /*                      hex/dec equivalent.                     **/
18  /*****************************************************************/
19
20  move.l #0x43000000, %a2      /* address of values to convert */
21  move.l #0x43100000, %a3      /* adresss of converted values to be stored */
22  move.l #100, %d7             /* amount of iterations for loop */
23
24
25  /* repeat: main loop. Checks each input and converts if possible. */
26  repeat:
27  move.l (%a2), %d2            /* move value from address into data register */
28  move.l %d2, (%a3)            /* move value in data register into address */
29  cmpi.l #0x0d, %d2            /* enter key pressed => exit */
30  beq done                     /* exits if enter key was pressed */
31
32  cmpi.l #0x30, %d2            /* compares the value to hex '0' */
33  blt error                    /* not a valid character if less than 0 */
34  cmpi.l #0x39, %d2            /* compare the value to hex '9' */
35  bgt higher                   /* go to higher to keep testing */
36
37  move.l #0x30, %d2            /* move value into data register */
38  sub.l %d2, (%a3)             /* subtract data register value from address value */
39  bra check                    /* check if done iterating */
40
41
42  /* higher: checks if it's a uppercase ascii character */
43  higher:
44  cmpi.l #0x41, %d2            /* compare value to hex 'A' */
45  blt error                    /* not a valid character; go to error */
46  cmpi.l #0x46, %d2            /* compare the value to hex 'F' */
47  bgt lower                    /* go to lower to keep testing */
48
49  move.l #0x37, %d2            /* move value into data register */
50  sub.l %d2, (%a3)             /* subtract value from d2 to value in address */
51  bra check                    /* go to check */
52
53
54  /* lower: checks if it's a lowercase ascii character */
55  lower:
56  cmpi.l #0x61, %d2            /* compare the value to hex 'a' */
57  blt error                    /* not a valid character; go to error */
58  cmpi.l #0x66, %d2            /* compare the value to hex 'f' */
```

```
59  bgt error                       /* go to continue to keep testing */
60  move.l #0x57, %d2               /* move value to data register */
61  sub.l %d2, (%a3)                /* subtract d2 from value in address */
62  bra check                       /* go to check */
63
64
65  /* check: checks if done iterating */
66  check:
67  add.l #4, %a2                   /* increment address by one long word */
68  add.l #4, %a3                   /* increment address by one long word */
69  sub.l #1, %d7                   /* subtract 1 from data register */
70  cmp.l #0, %d7                   /* see if 100 iterations are done */
71  beq done                        /* if yes, exit loop and go to done */
72  bra repeat                      /* else repeat loop */
73
74
75  /* error: moves error code to memory location */
76  error:
77  move.l #0xFFFFFFFF, (%a3)       /* move error code into memory location */
78  add.l #4, %a2                   /* increment address by one long word */
79  add.l #4, %a3                   /* increment address by one long word */
80  sub.l #1, %d7                   /* subtract value from data register */
81  cmp.l #0, %d7                   /* compare value with data register*/
82  beq done                        /* if equal, exit loop and go to done */
83  bra repeat                      /* else repeat loop */
84
85
86  /* done: exit point of program */
87  done:
88
89
90  /*End of program ************************************************/
91
92  /* DO NOT MODIFY THIS ------------------------------------------*/
93  movem.l (%a7),%d2-%d7/%a2-%a5 /*Restore data and address registers */
94  lea       40(%a7),%a7
95  rts
96  /*------------------------------------------------------------------*/
```

6.4 Part B Assembler Code

```
1  /* DO NOT MODIFY THIS ------------------------------------------*/
2  .text
3
4  .global AssemblyProgram
```

```
5
6  AssemblyProgram:
7  lea       -40(%a7),%a7 /*Backing up data and address registers */
8  movem.l %d2-%d7/%a2-%a5,(%a7)
9  /*-------------------------------------------------------------------*/
10
11 /*****************************************************************/
12 /* General Information ********************************************/
13 /* File Name: Lab1b.s *********************************************/
14 /* Names of Students: Lora Ma and Benjamin Kong                   **/
15 /* Date: 7 February 2020                                          **/
16 /* General Description:                                           **/
17 /* Convert ASCII char from uppercase to lowercase and lowercase   **/
18 /*   to uppercase                                                 **/
19 /*****************************************************************/
20
21 move.l #0x43000000, %a2      /* address of values to convert */
22 move.l #0x43200000, %a3      /* address of converted values to be stored */
23 move.l #100, %d7             /* amount of iterations for loop */
24
25
26 /* repeat: main loop. Checks each input and converts if possible. */
27 repeat:
28 move.l (%a2), %d2            /* move value from address into data register */
29 move.l %d2, (%a3)            /* move value in data register into address */
30 cmpi.l #0x0d, %d2            /* enter key pressed => exit */
31 beq done                     /* exits if enter key was pressed */
32
33 cmpi.l #0x41, %d2            /* compare the value to hex 'A' */
34 blt error                    /* not a valid character */
35
36 cmpi.l #0x5A, %d2            /* compare the value to hex 'Z' */
37 bgt lower                    /* go to lower */
38
39 move.l #0x20, %d2            /* move value to data register */
40 add.l %d2, (%a3)             /* add value in data register to address */
41 bra check                    /* check if done iterating */
42
43
44 /* lower: checks if it's a lowercase ascii character */
45 lower:
46 cmpi.l #0x61, %d2            /*compare the value to hex 'a' */
47 blt error                    /* not a valid character; go to error */
48 cmpi.l #0x7A, %d2            /* compare the value to hex 'z' */
49 bgt error                    /* go to lower*/
```

```
50  move.l #0x20, %d2            /*move value to data register*/
51  sub.l %d2, (%a3)             /* subtract data register and address value */
52  bra check                    /* go to check */
53
54
55  /* check: checks if done iterating */
56  check:
57  add.l #4, %a2                /* increment address by one long word */
58  add.l #4, %a3                /* increment address by one long word */
59  sub.l #1, %d7                /* subtract value from data register */
60  cmp.l #0, %d7                /* compare value with data register */
61  beq done                     /* if equal, exit loop and go to done */
62  bra repeat                   /* else repeat loop */
63
64
65  /* error: moves error code to memory location */
66  error:
67  move.l #0xFFFFFFFF, (%a3)    /* move error code into memory location */
68  add.l #4, %a2                /* increment address by one long word */
69  add.l #4, %a3                /* increment address by one long word */
70  sub.l #1, %d7                /* subtract value from data register */
71  cmp.l #0, %d7                /* compare value with data register */
72  beq done                     /* if equal, exit loop and go to done */
73  bra repeat                   /* else repeat loop */
74
75
76  /* done: exit point of program */
77  done:
78
79
80  /*End of program ***********************************************/
81
82  /* DO NOT MODIFY THIS ------------------------------------------*/
83  movem.l (%a7),%d2-%d7/%a2-%a5 /*Restore data and address registers */
84  lea      40(%a7),%a7
85  rts
86  /*-----------------------------------------------------------------*/
```